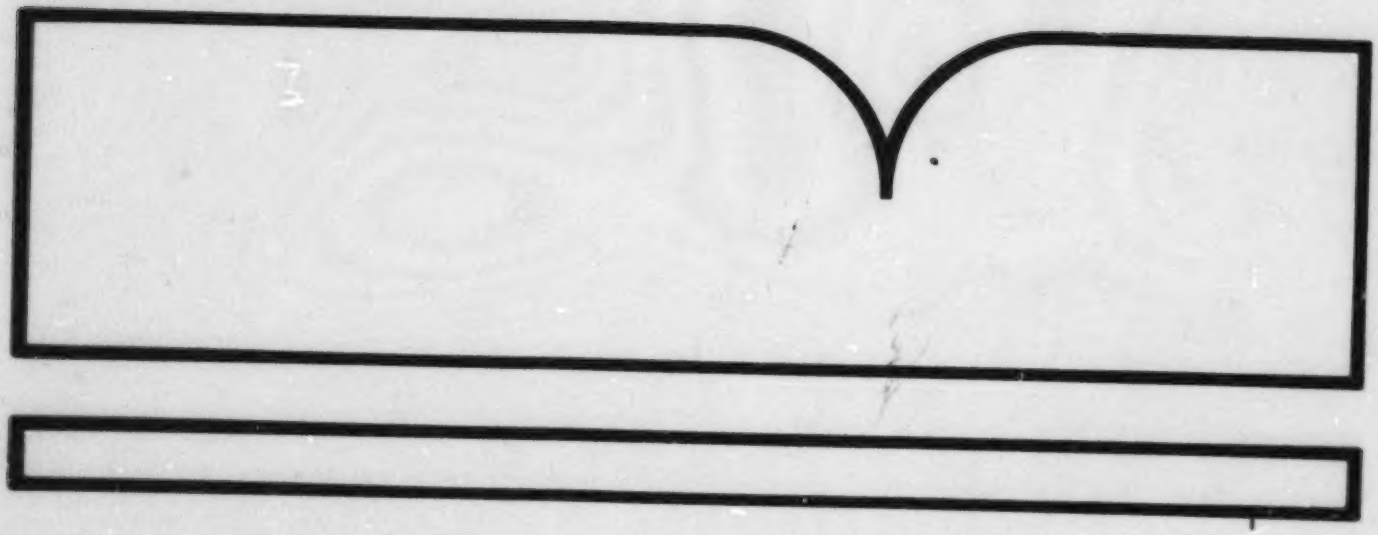
PB84-229905

Low-Cost Digitizer for the Apple

(U.S.) Environmental Research Lab.
Narragansett, RI

Jun 82



U.S. Department of Commerce
National Technical Information Service
NTIS

TECHNICAL REPORT DATA
(Please read Instructions on the reverse before completing)

1. REPORT NO.	2.	PB84-229905
EPA-600/J-82-415		

4. TITLE AND SUBTITLE	5. REPORT DATE
Low-Cost Digitizer for the Apple (Journal Version)	1982
	6. PERFORMING ORGANIZATION CODE
7. AUTHOR(S)	8. PERFORMING ORGANIZATION REPORT NO.
Jay Sinnett	ERLN-539
9. PERFORMING ORGANIZATION NAME AND ADDRESS	10. PROGRAM ELEMENT NO.
U. S. EPA, ERL, Narragansett South Ferry Road Narragansett, RI 02882	ARBIA
	11. CONTRACT/GRANT NO.
12. SPONSORING AGENCY NAME AND ADDRESS	13. TYPE OF REPORT AND PERIOD COVERED
U. S. EPA, ERL, Narragansett South Ferry Road Narragansett, RI 02882	Journal Article
	14. SPONSORING AGENCY CODE
	EPA/600/05

15. SUPPLEMENTARY NOTES

MICRO No. 49, June 1982, pp. 40-49.

16. ABSTRACT

If you have a computer with high-resolution graphics, then you already own 99% of a digitizer with equal resolution. This article tells you how to use a sheet of half-reflecting plastic to build the rest. The modular demonstration program should be easy to customize to your application. The article also describes a "crashproof" disk access which should be of interest to those who have puzzled over details of the Applesoft ONERR GOTO command.

17. KEY WORDS AND DOCUMENT ANALYSIS

a. DESCRIPTORS	b. IDENTIFIERS/OPEN ENDED TERMS	c. COSATI Field/Group

REPRODUCED BY
NATIONAL TECHNICAL
INFORMATION SERVICE
U.S. DEPARTMENT OF COMMERCE
SPRINGFIELD, VA. 22161

18. DISTRIBUTION STATEMENT	19. SECURITY CLASS *(This Report)*	21. NO. OF PAGES
Unlimited: Released to Public	Unclassfied	8
	20. SECURITY CLASS *(This page)*	22. PRICE
	Unclassified	

EPA Form 2220-1 (9-73)

EPA-600/J-82-415
JOURNAL ARTICLE

A Low-Cost Digitizer for the Apple

by Jay Sinnett

If you have a computer with high-resolution graphics, then you already own 99% of a digitizer with equal resolution. This article tells you how to use a sheet of half-reflecting plastic to build the rest. The modular demonstration program should be easy to customize to your application. The article also describes a "crashproof" disk access which should be of interest to those who have puzzled over details of the Applesoft ONERR GOTO command.

Digitizer Demo
requires:

Apple II or Apple II Plus
with 48K
One disk drive
Applesoft BASIC

Although **Digitizer Demo** is implemented on the Apple II, the method described here can be implemented on any microcomputer offering high-resolution graphics

A digitizer is a device used to translate pictures into sets of X and Y coordinates for use in a computer. Digitizers are used in research for measuring the size, shape, or position of objects that have been photographed or videotaped, reading strip chart recordings, etc. Hobbyists might use digitizers to copy shapes into the computer's memory from drawings made with pencil and paper. Unfortunately, most digitizers available for use with popular microcomputers cost several hundred dollars. In this article, I will describe one that can be built for under $20.00 and can be used with any microcomputer having high-resolution graphics. A sample program written for the Apple [1] demonstrates the simplicity of the software required to implement the digitizer.

Most inexpensive digitizers work this way: the operator moves a mechanical pointer around on a photo or drawing while its X and Y coordinates are measured electrically and sent to the computer. Usually the pointer has to be in physical contact with the picture so that the operator's perspective won't cause parallax errors. In this digitizer the pointer is actually a cursor on the Apple's high-resolution screen. This is optically superimposed onto the photo, drawing, or flat object to be digitized using a half-silvered ("two-way") mirror. When this is done correctly, there is no parallax error because both images are optically in the same place.

To build the digitizer, I first placed two identical monitors facing each other, about 10" apart, with a sheet of window-plastic exactly halfway between them and parallel to the monitor screens, as shown in figure 1. This worked, but plastic alone reflected weakly, and there was an image from both near and far sides, which was distracting. Then I obtained some half-silvered two-way mirror plastic film and applied it to the near side of the window: success! Both direct and reflected images were equally strong, and only one reflection was visible.

The reflecting plastic I used was secondhand and did not form a perfectly smooth surface. Because the uneveness

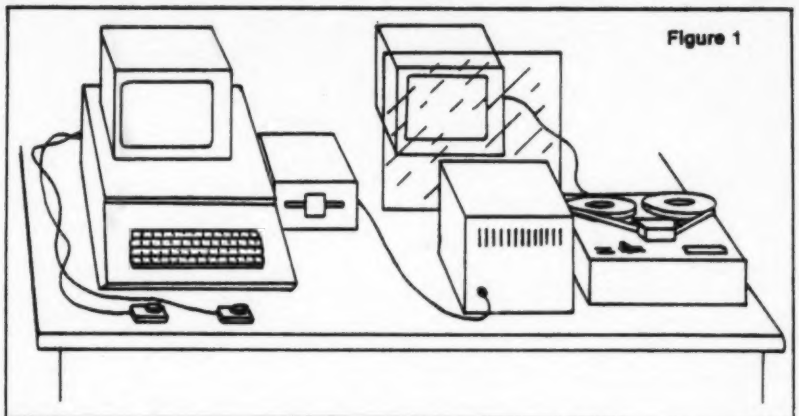


Figure 1

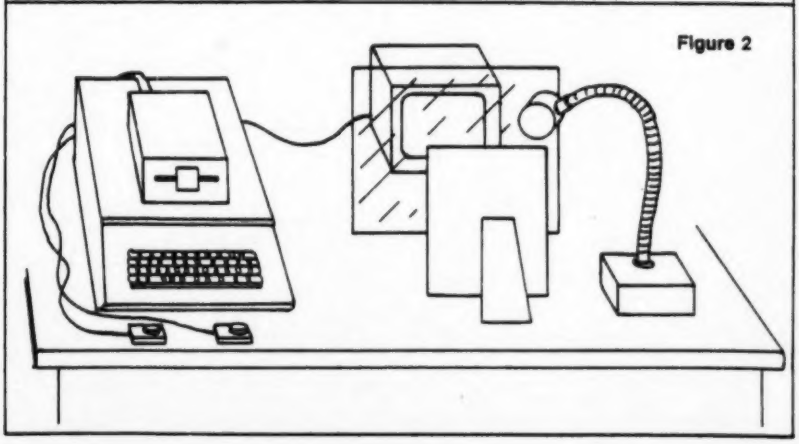


Figure 2

caused by tiny bubbles in the plastic distorted the reflected image, I found it best to look through the mirror at the video image, while using the reflected image of the Apple monitor. (A slight distortion of the cursor on the reflected Apple monitor seemed to be less distracting than distortion of the picture being digitized.) This arrangement also let me see the picture without a left-right reversal caused by mirror reflection. A third monitor was used for normal Apple monitor functions, so that the one being used for digitizing could be left in place once it was aligned.

If you want to digitize photos, drawings, etc., and you don't have extra monitors, I recommend you follow the layout of figure 2. In this setup you can read your Apple monitor normally through the glass. New one-way plastic carefully applied to a clean pane of window glass should give you acceptably clear reflections of the picture you are working on; if not, you can get a glass one-way mirror. The one-way plastic is sold in nationwide automotive parts and customizing shops; a 20-inch by 10-foot roll costs $10 - $15. One-way glass is more expensive; an 18-inch by 18-inch pane will set you back $20 -$30. A reading lamp may be needed to balance the intensities of the two images.

Before you can begin digitizing, the images must be aligned correctly, and the best tools are your eyes. First, set up the equipment as shown in figure 2. Be careful to have the mirror, monitor, and picture vertical, not leaning either way. Run the DIGITIZER DEMO program and use the paddles to align the cursor from the Apple monitor directly over the reflection of some spot on the picture. A small dot that just fits within the cursor works well. Now lean your head close to the mirror. With the aid of binocular (both eyes) depth perception, you should be able to see whether the cursor or the spot appears to be closer to you. Move either the picture or the monitor until the reflected image appears to be exactly the same distance from you as the non-reflected one. (This is easier than it sounds!) Check two or three points to be sure the angles are right.

With precise alignment, the image of a stationary cursor won't move with respect to the picture even when you move your head. In fact two people can look at the setup at once and see the same thing! The principle behind this position-independent superimposition is the optical law that states: "the angle of incidence is equal to the angle of reflection." This law will work even

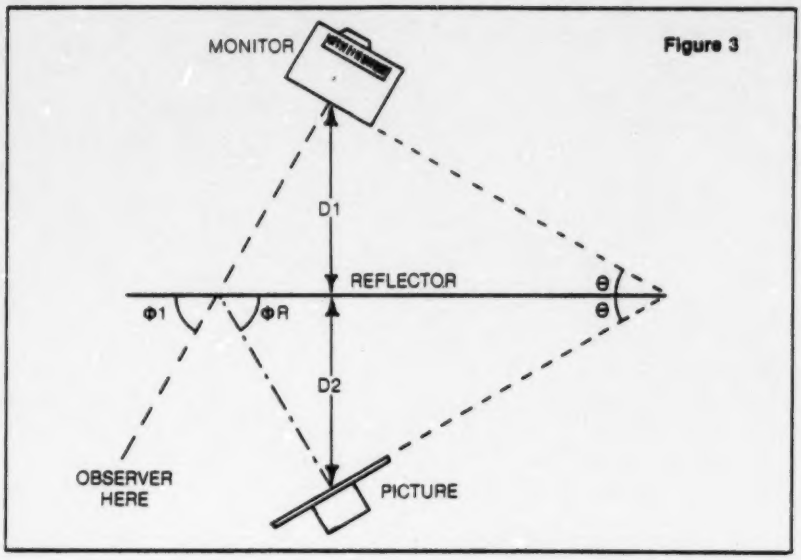


Figure 3

if the monitor screen and the picture are not parallel to the glass. Just be sure that the reflection lines up exactly with the direct image. In figure 3, as long as D1 = D2 and Θ is the same on both sides, the eye can be moved freely and the images will remain superimposed.

Obviously, the size of the picture you can digitize is limited by the size of your monitor screen, and the resolution is set by the Apple's 280 by 192 high-resolution grid. Most television monitors introduce some distortion to both horizontal and vertical axes, squeezing the picture in some places and stretching it in others. To remedy this, calibrate your own particular monitor by digitizing many points from an accurate target grid (such as graph paper). Then calculate correction factors to map the monitor's coordinates into an accurate array.

The digitizer demonstration program presented in listing 1 was designed to allow easy modification or expansion. It is written in structured format with several small blocks of code, each performing only one well-defined function, much like procedures in Pascal.

Lines	
1000-1240	Initialize variables.
1700-1850	Load a binary file which defines cursor shapes.
2000-2050	Read paddle values. Scale to be within the range 0-1.
3000-3070	Translate paddle values to X and Y coordinates for display. Use offset and scale factors.
4000-4080	Draw cursor on screen at selected location. Remember location.
4500-4550	Erase cursor from remembered screen location.
5000-5050	Print current cursor location on screen in text window.
6000-6090	Read and debounce buttons.
7000-7560	Toggle from coarse to fine motion control and back again.
8000-8050	Pseudo output routine... print cursor location on screen in text windows.
9000-9050	Check keyboard for any activity.
20000-20160	Main routine.

Lines 1060, 1070, and 1180 extract the address of a BLOADED shapefile from the DOS file pointer[2] (of a 48K Apple) and copy it into the Applesoft shapefile pointer. (This technique can be helpful in shape-generating programs, too.)

Lines 1700-1850 are a crashproof version of a BLOAD command that returns an error flag, ER, if the specified file is not on disk. If there is any kind of error in reading file SF$ from the disk in line 1780, control will immediately be passed to line 1800. This is the function of the Applesoft ONERR GOTO 1800 command in line 1770. The Applesoft manual[3] warns you that ONERR GOTO has a bug and gives you a machine-language patch to fix it. This is exercised by the CALL 768 in line

1840 (which assumes that the patch has been placed into memory as instructed in the manual). When the Apple returns from this call, the return address for the most recent unfinished GOSUB has been discarded. I use a dummy GOSUB in line 1730, so it can be thrown away if there is an error. A normal BLOAD will result in line 1790 RETURNING to line 1740 but the error routine ends with a GOTO 1740 instead of a RETURN. Both the normal and the error exits from the BLOAD routine execute lines 1740 and 1750.

Control of cursor motion can be accomplished in at least two ways. I have used the paddles for cursor motion (lines 1000-3070), but I have seen perfectly good cursor motion controls implemented on a PET with just the standard keyboard. In this program I provide both coarse and fine motion control for two reasons. First, the 256 possible values for the X-axis paddle (0) cannot be translated into 280 values for the screens X-axis without leaving out some points, whereas only 192 of the possible 256 Y-axis paddle (1) values are used. Second, it is easier to make fine position adjustments when the paddles are less sensitive and operate over a small section of the screen.

The mode is controlled by the button on paddle 0, which makes the program toggle back and forth between coarse and fine resolution every time it is pressed. The apparent center of the fine-tuning square is placed exactly where the coarse cursor was when the button was pressed, and different cursors are used for each mode so that the operator knows which mode he is using. The button on paddle 1 triggers an output subroutine that could send a value to disk or tape (but is displayed on the screen in this demonstration).

There are no IF statements in the routines that draw and erase the cursor, and the only IF statements in the paddle-to-screen translation are for range checking. The coarse/fine switching is all accomplished by resetting variables for screen origin (XO,YO), scale factor (XM,YM), paddle shift (FT), and cursor shape (CU) (lines 7000-7560). I have found this technique of modifying variables to set flags and to pass parameters is much easier to write and debug than a maze of IF statements. It forces you to plan your program before you start writing, but the effort is repaid many times over.

Whenever I plan to use high-resolution graphics, I first reset the Apple's memory pointers to protect the

Listing 1

```
10  GOTO 20000: REM MAIN PROGRAM (STRUCTURED PROGRAM HAS SUBROUTINES FIRS
    T)
20  REM ***************
30  REM
40  REM     DIGITIZER DEMO REV 1
50  REM
60  REM  BY JAY C. SIDNETT
70  REM          JULY 8, 1981
80  REM  U.S. EPA ENVIRONMENTAL RESEARCH LABORATORY
90  REM  SOUTH FERRY ROAD
100  REM NARRAGANSETT, RI 02882
110  REM
120  REM ***************
1000  REM
1010  REM       INITIALIZATION ROUTINE
1020  REM
1030  TEXT : HOME
1040 CD$ =  CHR$ (4): REM CONTROL-D
1050 KA = 49152: REM KEYBOARD ADDRESS
1060 BA = 43634: REM ADDRESS OF DOS POINTER TO BLOADED FILE (48K APPLE)
1070 PL = 232: REM POINTER TO SHAPE TABLE USED BY APPLESOFT
1080 BO = 49249:B1 = 49250: REM ADDRESSES FOR GAME CONTROL BUTTONS
1090 XO = 0:YO = 0: REM OFFSET (FLOATING ORIGIN)
1100 XH = 279:YH = 191: REM HIGHEST VALUES FOR SCREEN X,Y
1110 XF = 64:YF = 64: REM SCALE FACTOR FOR FINE MOTION
1120 XM = XH:YM = YH: REM MULTIPLIER (EITHER FULL SCREEN OR FINE MOTION)
1130  PRINT "WHAT IS THE NAME OF YOUR CURSOR SHAPE TABLE ";
1140  INPUT "(RETURN FOR SUCCURSOR)? ";FI$
1150 FI = 0:SP$ = "SUCCURSOR": IF  LEN (FI$) <  > 0 THEN SP$ = FI$:FI = 1

1160  GOSUB 1700: REM SAFE BLOAD
1170  IF ER THEN  PRINT "SORRY, CAN'T FIND "SP$: PRINT CD$"CATALOG": GOTO
    1130
1180  POKE PL, PEEK (BA): POKE PL + 1, PEEK (BA + 1): REM SET SHAPETABLE
    POINTER
1190 CC = 2:CF = 3: IF  NOT FI THEN 1220: REM DEFAULT IS SUCCURSOR SET
1200  INPUT "WHICH CURSOR NUMBER DO YOU WANT TO USE  FOR COARSE POSITION?
    ";CC
1210  INPUT "WHICH CURSOR DO YOU WANT FOR FINE       POSITIONING? ";CF
1220 CU = CC: REM START WITH COARSE CURSOR
1230  HOME : SCALE= 1: ROT= 0: HGR
1240  RETURN
1700  REM
1710  REM       BLOAD SP$
1720  REM
1730  GOSUB 1760: REM DUMMY CALL FOR ERROR TRAP TO WORK
1740  POKE 216,0: REM TURN OFF ERROR PROCESSING
1750  RETURN : REM TO MAIN CALLING ROUTINE
1760 ER = 0: REM ERROR FLAG RETURNED TO CALLING ROUTINE
1770  ONERR  GOTO 1800
1780  PRINT CD$"BLOAD"SP$
1790  RETURN : REM SUCCESSFUL
1800  REM
1810  REM     ERROR TRYING TO LOAD FILE
1820  REM
1830 ER =  PEEK (222): REM ERROR CODE, IF YOU WANT
1840  CALL 768: REM STRAIGHTEN OUT SUBROUTINE STACK, REMOVING ONE RETURN
    ADDRESS
1850  GOTO 1740
2000  REM
2010  REM       READ PADDLES, SCALE 0 TO 1
2020  REM
2030 XP =  PDL (0) / 255
2040  FOR N = 1 TO 10: NEXT : REM DELAY FOR PDL CIRCUIT
2050 YP =  PDL (1) / 255
3000  REM
3010  REM     TRANSLATE PDL VALUES
3020  REM
3030 X = XO +  INT ((XP - FT) * XM): IF X < 0 THEN X = 0
3040  IF X > XH THEN X = XH
3050 Y = YO +  INT ((YP - FT) * YM): IF Y < 0 THEN Y = 0
3060  IF Y > YH THEN Y = YH
3070  RETURN
4000  REM
4010  REM     PLACE CURSOR ON SCREEN
4020  REM
4030 XS = XH - X: REM REVERSE LEFT/RIGHT
4040 YS = YH - Y: REM ORIGIN AT BOTTOM OF SCREEN
4050  HCOLOR= 3
4060  DRAW CU AT XS,YS
4070 X1 = X:Y1 = Y: REM REMEMBER COORDINATES USED LAST
4080  RETURN
4500  REM
4510  REM     ERASE CURSOR FROM SCREEN
4520  REM
4530  HCOLOR= 0
4540  DRAW CU AT (XH - X1),(YH - Y1)
4550  RETURN
5000  REM
5010  REM     DISPLAY CURRENT CURSOR LOCATION
```

(Continued)

Listing 1 *(Continued)*

```
5020  REM
5030  VTAB 22: PRINT "                                        ": REM ERASE LAST
      DATA
5040  VTAB 22: PRINT "X = "X"  Y = "Y
5050  RETURN
6000  REM
6010  REM      READ BUTTONS (DEBOUNCING)
6020  REM
6030 F0 = 0:F1 = 0
6040 SA =  PEEK (S0) > 127: REM 1 IF BUTTON 0 DOWN
6050 SB =  PEEK (S1) > 127: REM BUTTON 1
6060  IF SA AND  NOT SQ THEN F0 = 1: REM F0 IS FLAG FOR BUTTON 0 BEING PR
      ESSED
6070  IF SB AND  NOT SR THEN F1 = 1: REM FLAG FOR BUTTON 1
6080 SQ = SA:SR = SB: REM REMEMBER PREVIOUS STATE
6090  RETURN
7000  REM
7010  REM     TOGGLE SCALE FACTOR, ORIGIN
7020  REM
7030  IF XM = XH THEN 7500
7040  REM ELSE HERE TO CHANGE FROM FINE TO COARSE
7050 XO = 0:YO = 0: REM ORIGIN
7060 XM = XH:YM = YH: REM SCALE FACTOR
7070 CU = CC: REM CURSOR TYPE
7080 FT = 0: REM NO SHIFT OF PADDLE
7090  GOTO 7560 REM END ELSE
7500  REM HERE TO CHANGE FROM COARSE TO FINE MOTION
7510 XO = X
7520 YO = Y
7530 XM = XF:YM = YF: REM  SCALE
7540 CU = CF: REM CURSOR
7550 FT = .5: REM SHIFT PADDLES TO CENTERED PLUS/MINUS RANGE
7560  RETURN
8000  REM
8010  REM     OUTPUT ROUTINE...TO SCREEN FOR THIS DEMO
8020  REM
8030  VTAB 23: PRINT "                                          ": REM ERASE
      OLD POINT
8040  VTAB 23: PRINT "LAST 'PLOTTED' POINT: X = "X"  Y = "Y
8050  RETURN
9000  REM
9010  REM     CHECK KEYBOARD
9020  REM
9030 KF =  PEEK (KA) > 127: REM TRUE IF A KEY HAS BEEN PRESSED
9040  IF KF THEN  GET KB$: REM RETURN WITH IDENTITY OF KEY IN KB$
9050  RETURN
20000  REM
20010  REM       MAIN ROUTINE
20020  REM
20030  GOSUB 1000: REM INITIALIZE
20040  GOSUB 2000: REM GET DESIRED COORDINATES
20050  IF X1 = X AND Y1 = Y THEN  GOTO 20110
20060  REM IF COORDINATES HAVE CHANGED, ERASE OLD CURSOR AND PLACE NEW ON
      E
20070  GOSUB 4500: REM ERASE OLD CURSOR
20080  GOSUB 4000: REM PLACE NEW CURSOR
20090  GOSUB 5000: REM DISPLAY CURRENT LOCATION
20100  REM END IF
20110  GOSUB 6000: REM READ BUTTONS INTO F0, F1
20120  IF F0 THEN  GOSUB 4500: GOSUB 7000: GOSUB 4000: REM TOGGLE SCALE F
      ACTOR
20130  IF F1 THEN  GOSUB 8000: REM 'OUTPUT' A DATA POINT
20140  GOSUB 9000: REM CHECK KEYBOARD
20150  IF KF THEN  END : REM FOR THIS DEMO
20160  GOTO 20040
```

Listing 2

```
800.882

0800- 03 00 08 00 1F 00 50 00
0808- 24 24 24 B4 92 D2 3F 3F
0810- 3F 4D 49 09 2D 2D 2D DF
0818- DB 13 36 36 16 24 00 DB
0820- 3F 3F 24 64 0D 18 0C 0D
0828- 18 2D 35 36 26 08 18 08
0830- 2D 0E 0E 0E 0E 36 FE 1B
0838- 2D AD 36 1E 1E 1E 1E 3F
0840- 27 24 B4 D2 3F 0F 18 0F
0848- 18 0F 18 0F 18 24 04 00
0850- 1B 3F 3F 6F 20 64 0D 18
0858- 2D 25 30 16 36 0C 18 08
0860- 2D 0E 0E 36 FE 2B 6D FD
0868- 8B 36 CE F3 1E 38 0F 18
0870- 08 18 36 36 26 C8 3B 0F
0878- 18 0F 18 0F 98 51 E2 1C
0880- 24 04 00
*
```

entire space from $800 to the top of page one high-resolution graphics at $4000. This leaves about 20K of contiguous program space and 6K of shapefile space available. The following series of POKEs should be used to accomplish this:

```
POKE 103,1
POKE 104,64
POKE 16384,0
```

The shape table used by this program is shown in monitor format (listing 2), occupying locations $800-$882. It actually has three cursors available: a crosshair and two different circle/crosshair combinations. The DIGITIZER DEMO program assumes that it is stored on disk under the name BUGCURSOR. If you use it as shown, you *must* reset the memory pointers before running DIGITIZER DEMO, or the program will be destroyed when the shapefile is loaded.

The hardware and software I have described here were originally developed for a government research project, but they should be useful for a great many other applications. Your output routine might simply connect points as you enter them, making a line drawing, or you might want to save points in a shapefile or in a disk data file. All sorts of graphics programming should be easier and more fun with this affordable digitizer.

References

1. Mention of commercial products does not imply endorsement by the U.S. Environmental Protection Agency.

2. *The DOS Manual*, Apple Computer, Inc., Cupertino, CA 95014, 1980. Page 144 (DOS Entry Points).

3. *Basic Programming Reference Manual*, Apple Computer, Inc., Cupertino, CA 95014, 1978. Page 136 (Machine-Language program to clear up ONERR GOTO problem).

Jay Sinnett has a B.S. degree in electronics from MIT (1968). After three years in the Navy, he joined the Environmental Protection Agency and has been at his present position for seven years. He programs extensively in machine language and Microsoft BASIC on the SYM, PET, and Apple II at work. He may be contacted at the United States Environmental Protection Agency, Environmental Research Laboratory, South Ferry Road, Narragansett, RI 02882.